```lua
---- ---- ---- ---- General Functions
local l={}

---- Cache names known `b4` we start
-- Use that, later, to hunt down any rogue globals.
l.b4={}; for k,_ in pairs(_ENV) do l.b4[k]=k end
function l.rogues()
  for k,v in pairs(_ENV) do if not l.b4[k] then print("?",k,type(v)) end end end

---- Classes
function l.klass(sName,     new,self,t)
  function new(k,...)
    self = setmetatable({},k)
    return setmetatable(k.new(self,...) or self,k) end
  t={_is = sName, __tostring = l.cat}
  t.__index = t
  return setmetatable(t,{__call=new}) end

---- ---- ---- Misc
-- Do nothing.
function l.same(x) return x end

---- ---- ---- Maths
-- Large number
l.big=math.huge

-- Random num
l.rand=math.random

-- Round nums.
function l.rnd(num, places)
  local mult = 10^(places or 3)
  return math.floor(num * mult + 0.5) / mult end

---- ---- ---- Lists
-- Return any item (selected at random) from list `t`.
function l.any(t) return t[l.rand(#t)] end

-- Return `num` items (selected at random) from list `t`.
-- If `num` is more than the size of the list, return that list, shuffled.
function l.many(t,num, u)
  if num>#t then return l.shuffle(t) end
  u={}; for j=1,num do u[1+#u]= l.any(t) end; return u end

-- Return items in `t` filtered through `f`. If `f` ever returns nil
-- then the returned list will be shorter.
function l.map(t,f)
  local u={}; for _,v in pairs(t) do u[1+#u]=f(v) end; return u end

-- Helper function for `map` (extracts certain slots
function l.get(x) return function(t) return t[x] end end

-- Return the `p`-th item in `t` (assumed to be sorted). e.g.
-- `per(t,.5)` returns the median.
function l.per(t,p)
  p=math.floor((p*#t)+.5); return t[math.max(1,math.min(#t,p))] end

-- Add `x` to list `t`, returning `x`.
function l.push(t,x) t[1+#t]=x; return x end

-- In-place reverse, return reversed list
function l.rev(t)
  for i=1, math.floor(#t / 2) do t[i],t[#t-i+1] = t[#t-i+1],t[i] end
  return t end

-- Randomly shuffle, in place, the list `t`.
function l.shuffle(t,   j)
  for i=#t,2,-1 do j=rand(i); t[i],t[j]=t[j],t[i] end; return t end

-- Return `t` from `go` to `stop` by `inc`.
-- `go` is optional (defaults to 1).
-- `stop` is optional (defaults to length of `t`).
-- `inc` is optional (defaults to 1)
function l.slice(t, go, stop, inc)
  local u={}
  for j=(go or 1)//1,(stop or #t)//1,(inc or 1)//1 do u[1+#u]=t[j] end
  return u end

---- ---- ---- Sorting
-- Sorting predictates
function l.gt(x) return function(a,b) return a[x] > b[x] end end
function l.lt(x) return function(a,b) return a[x] < b[x] end end

-- In-place sort, returns sorted list
function l.sort(t,f) if #t==0 then t=l.values(t) end; table.sort(t,f); return t end

-- Return values in a table
function l.values(t,   u) u={}; for _,v in pairs(t) do u[1+#u]=v end; return u end

---- ---- ---- Print
-- Generate a string from `t`.
function l.cat(t,   seen,      show,u,pub)
  if type(t)~="table" then return tostring(t) end
  seen = seen or {}
  if seen[t] then return "..." end
  seen[t]=t
  function show(k,v)
    if tostring(k):sub(1,1) ~= "_" then
      v=l.cat(v,seen)
      return #t==0 and l.fmt(":%s %s",k,v) or tostring(v) end end
  u={}; for k,v in pairs(t) do u[1+#u]=show(k,v) end
  return (t._is or "").."{"..table.concat(#t==0 and l.sort(u) or u," ").."}" end

-- Generate a string from `t` and print it (returning `t`).
function l.chat(t) print(l.cat(t)) return t end

-- Emulate Printf
l.fmt = string.format

---- ---- ---- Read
-- Try reading `str` as a boolean, then int, then float, then string.
function l.coerce(str)
  str = str:match"^%s*(.-)%s*$"
  if str=="true" then return true elseif str=="false" then return false
  else return math.tointeger(str) or tonumber(str) or str end  end

-- Read update for `slot` of table from command line flag `-s` or `--slot`.
-- If slot's is a boolean, this code flips old value.
function l.cli(t)
  for slot,v in pairs(t) do
    v = tostring(v)
    for n,x in ipairs(arg) do
      if x=="-"..(slot:sub(1,1)) or x=="--"..slot then
        v = v=="false" and "true" or v=="true" and "false" or arg[n+1] end end
    t[slot] =  l.coerce(v) end
  return t end

-- Read lines from `filestr`, converting each into words, passing that to `fun`.
function l.csv(filename, fun)
  l.lines(filename, function(t) fun(l.words(t,",",l.coerce)) end) end

--- Read lines from `filestr`, closing stream at end. Call `fun` on each line.
function l.lines(filename, fun)
  local src = io.input(filename)
  while true do
    local str = io.read()
    if not str then return io.close(src) else fun(str) end end end

-- Split  `str` on `sep`, filter each part through `fun`, return the resulting list.
function l.words(str,sep,fun,      t)
  fun = fun or function(z) return z end
  sep = l.fmt("([^%s]+)",sep)
  t={};for x in str:gmatch(sep) do t[1+#t]=fun(x) end;return t end

-- Update settings from command line. Run start up functions.
-- Before running one function,   reset random number seed. Afterwards,
-- reset settings to whatever they were before the action.
function l.main(sHelp,settings,funs)
  settings = l.cli(settings)
  if settings.help
  then os.exit(print(sHelp:gsub("[%u][%u%d]+","\27[1;36m%1\27[0m")
                           :gsub("([-][%S]+)","\27[1;33m%1\27[0m"),"")) end
  local fails=0
  local saved={};for slot,value in pairs(settings) do saved[slot]=value end
  local todo ={}; for k,_ in pairs(funs) do l.push(todo,k) end -- Run tests.
  todo = settings.go=="all" and l.sort(todo) or {settings.go}
  for _,str in pairs(todo) do
    if   type(funs[str]) ~= "function"
    then return print("?? unknown startup action",str)
    else math.randomseed(settings.seed)
         if true ~= funs[str]() then fails=fails+1; print("FAIL",str) end
         for slot,value in pairs(saved) do settings[slot]=value end end
  end ------
  l.rogues()       -- Check for rogue local.
  os.exit(fails) end -- Report failures were seen.i

---- ---- ---- ---- Return
-- That's all folks.
return l
```